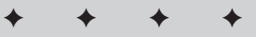
CHAPTER 1

An Eagle's Eye View of XML



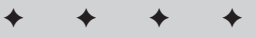
In This Chapter

What is XML?

Why are developers excited about XML?

The life of an XML document

Related technologies



This chapter introduces you to XML, the Extensible Markup Language. It explains, in general terms, what XML is and how it is used. It shows you how different XML technologies work together, and how to create an XML document and deliver it to readers.

What Is XML?

XML stands for Extensible Markup Language (often miscapitalized as *eXtensible Markup Language* to justify the acronym). XML is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. It is a meta-markup language that defines a syntax in which other domain-specific markup languages can be written.

XML is a meta-markup language

The first thing you need to understand about XML is that it isn't just another markup language like HTML, TeX, or troff. These languages define a fixed set of tags that describe a fixed number of elements. If the markup language you use doesn't contain the tag you need, you're out of luck. You can wait for the next version of the markup language, hoping that it includes the tag you need; but then you're really at the mercy of whatever the vendor chooses to include.

XML, however, is a meta-markup language. It's a language that lets you make up the tags you need as you go along. These tags must be organized according to certain general principles, but they're quite flexible in their meaning. For example, if you're working on genealogy and need to describe family names, personal names, dates, births, adoptions, deaths, burial sites, marriages, divorces, and so on, you can create tags for each of these. You don't have to force your data to fit into paragraphs, list items, table cells, or other very general categories.

You can document the tags you create in a schema written in any of several languages, including document type definitions (DTDs) and the W3C XML Schema Language. You'll learn more about DTDs and schemas in Parts II and IV of this book. For now, think of a schema as a vocabulary and a syntax for certain kinds of documents. For example, the schema for Peter Murray-Rust's Chemical Markup Language (CML) is a DTD that describes a vocabulary and a syntax for the molecular sciences: chemistry, crystallography, solid-state physics, and the like. It includes tags for atoms, molecules, bonds, spectra, and so on. Many different people in the field can share this schema. Other schemas are available for other fields, and you can create your own.

XML defines the meta syntax that domain-specific markup languages such as MusicXML, MathML, and CML must follow. It specifies the rules for the low-level syntax, saying how markup is distinguished from content, how attributes are attached to elements, and so forth, without saying what these tags, elements, and attributes are or what they mean. It gives the patterns that elements must follow without specifying the names of the elements. For example, XML says that tags begin with a < and end with a >. However, XML does not tell you what names must go between the < and the >.

If an application understands this meta syntax, it at least partially understands all the languages built from this meta syntax. A browser does not need to know in advance each and every tag that might be used by thousands of different markup languages. Instead, the browser discovers the tags used by any given document as it reads the document or its schema. The detailed instructions about how to display the content of these tags are provided in a separate style sheet that is attached to the document.

For example, consider the three-dimensional Schrödinger equation:

$$i\hbar\frac{\partial\psi(\boldsymbol{r}, t)}{\partial t} = -\frac{\hbar^2}{2m}\nabla^2\psi(\boldsymbol{r}, t) + V(\boldsymbol{r})\psi(\boldsymbol{r}, t)$$

XML means you don't have to wait for browser vendors to catch up with your ideas. You can invent the tags you need, when you need them, and tell the browsers how to display these tags.

XML describes structure and semantics, not formatting

XML markup describes a document's structure and meaning. It does not describe the formatting of the elements on the page. You can add formatting to a document with a style sheet. The document itself only contains tags that say what is in the document, not what the document looks like.

By contrast, HTML encompasses formatting, structural, and semantic markup. <B> is a formatting tag that makes its content bold. <STRONG> is a semantic tag that means its contents are especially important. <TD> is a structural tag that indicates that the contents are a cell in a table. In fact, some tags can have all three kinds of meaning. An <H1> tag can simultaneously mean 20-point Helvetica bold, a level 1 heading, and the title of the page.

For example, in HTML, a song might be described using a definition title, definition data, an unordered list, and list items. But none of these elements actually have anything to do with music. The HTML might look something like this:

```
<DT>Hot Cop
<DD> by Jacques Morali, Henri Belolo, and Victor Willis
<UL>
<LI> Jacques Morali
<LI> PolyGram Records
<LI> 6:20
<LI> 1978
<LI> Village People
</UL>
```

In XML, the same data could be marked up like this:

```
<SONG>
  <TITLE>Hot Cop</TITLE>
  <COMPOSER>Jacques Morali</COMPOSER>
  <COMPOSER>Henri Belolo</COMPOSER>
  <COMPOSER>Victor Willis</COMPOSER>
  <PRODUCER>Jacques Morali</PRODUCER>
  <PUBLISHER>PolyGram Records</PUBLISHER>
  <LENGTH>6:20</LENGTH>
  <YEAR>1978</YEAR>
  <ARTIST>Village People</ARTIST>
</SONG>
```

Instead of generic tags such as <DT> and <LI>, this example uses meaningful tags such as <SONG>, <TITLE>, <COMPOSER>, and <YEAR>. These tags didn't come from any preexisting standard or specification. I just made them up on the spot because they fit the information I was describing. Domain-specific tagging has a number of advantages, not the least of which is that it's easier for a human to read the source code to determine what the author intended.

XML markup also makes it easier for nonhuman automated computer software to locate all of the songs in the document. A computer program reading HTML can't tell more than that an element is a `DT`. It cannot determine whether that `DT` represents a song title, a definition, or some designer's favorite means of indenting text. In fact, a single document might well contain `DT` elements with all three meanings.

XML element names can be chosen such that they have extra meaning in additional contexts. For example, they might be the field names of a database. XML is far more flexible and amenable to varied uses than HTML because a limited number of tags don't have to serve many different purposes. XML offers an infinite number of tags to fill an infinite number of needs.

Why Are Developers Excited About XML?

XML makes easy many web-development tasks that are extremely difficult with HTML, and it makes tasks that are impossible with HTML possible. Because XML is extensible, developers like it for many reasons. Which reasons most interest you depends on your individual needs, but once you learn XML, you're likely to discover that it's the solution to more than one problem you're already struggling with. This section investigates some of the generic uses of XML that excite developers. In Chapter 2, you'll see some of the specific applications that have already been developed with XML.

Domain-specific markup languages

XML enables individual professions (for example, music, chemistry, human resources) to develop their own domain-specific markup languages. Domain-specific markup languages enable practitioners in the field to trade notes, data, and information without worrying about whether or not the person on the receiving end has the particular proprietary payware that was used to create the data. They can even send documents to people outside the profession with a reasonable confidence that those who receive them will at least be able to view the documents.

Furthermore, creating separate markup languages for different domains does not lead to bloatware or unnecessary complexity for those outside the profession. You may not be interested in electrical engineering diagrams, but electrical engineers are. You may not need to include sheet music in your web pages, but composers do. XML lets the electrical engineers describe their circuits and the composers notate their scores, mostly without stepping on each other's toes. Neither field needs special support from browser manufacturers or complicated plug-ins, as is true today.

Self-describing data

Much computer data from the last 40 years is lost, not because of natural disaster or decaying backup media (though those are problems too, ones XML doesn't solve), but simply because no one bothered to document how the data formats. A Lotus 1-2-3 file on a 15-year-old 5.25-inch floppy disk might be irretrievable in most corporations today without a huge investment of time and resources. Data in a less-known binary format such as Lotus Jazz may be gone forever.

XML is, at a low level, an incredibly simple data format. It can be written in 100 percent pure ASCII or Unicode text, as well as in a few other well-defined formats. Text is reasonably resistant to corruption. The removal of bytes or even large sequences of bytes does not noticeably corrupt the remaining text. This starkly contrasts with many other formats, such as compressed data or serialized Java objects, in which the corruption or loss of even a single byte can render the rest of the file unreadable.

At a higher level, XML is self-describing. Suppose you're an information archaeologist in the twenty-third century and you encounter this chunk of XML code on an old floppy disk that has survived the ravages of time:

```
<PERSON ID="p1100" SEX="M">
  <NAME>
    <GIVEN>Judson</GIVEN>
    <SURNAME> McDaniel</SURNAME>
  </NAME>
  <BIRTH>
    <DATE>21 Feb 1834</DATE>  </BIRTH>
  <DEATH>
    <DATE>9 Dec 1905</DATE>  </DEATH>
</PERSON>
```

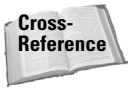
Even if you're not familiar with XML, assuming you speak a reasonable facsimile of twentieth-century English, you've got a pretty good idea that this fragment describes a man named Judson McDaniel, who was born on February 21, 1834 and died on December 9, 1905. In fact, even with gaps in or corruption of the data, you could probably still extract most of this information. The same could not be said for a proprietary, binary spreadsheet or word-processor format.

Furthermore, XML is very well documented. The World Wide Web Consortium (W3C)'s XML specification and numerous books tell you exactly how to read XML data. There are no secrets waiting to trip the unwary.

Interchange of data among applications

Because XML is nonproprietary and easy to read and write, it's an excellent format for the interchange of data among different applications. XML is not encumbered by copyright, patent, trade secret, or any other sort of intellectual property restrictions. It has been designed to be extremely expressive and very well structured while at the same time being easy for both human beings and computer programs to read and write. Thus, it's an obvious choice for exchange languages.

One such format is the Open Financial Exchange 2.0 (OFX, `http://www.ofx.net/`). OFX is designed to let personal finance programs, such as Microsoft Money and Quicken, trade data. The data can be sent back and forth between programs and exchanged with banks, brokerage houses, credit card companies, and the like.



OFX is further discussed in Chapter 2.

By choosing XML instead of a proprietary data format, you can use any tool that understands XML to work with your data. You can even use different tools for different purposes, one program to view and another to edit, for example. XML keeps you from getting locked into a particular program simply because that's what your data is already written in, or because that program's proprietary format is all your correspondent can accept.

For example, many publishers require submissions in Microsoft Word. This means that most authors have to use Word, even if they would rather use OpenOffice.org Writer or WordPerfect. This makes it extremely difficult for any other company to publish a competing word processor unless it can read and write Word files. To do so, the company's programmers must reverse-engineer the binary Word file format, which requires a significant investment of limited time and resources. Most other word processors have a limited ability to read and write Word files, but they generally lose track of graphics, macros, styles, revision marks, and other important features. Word's document format is undocumented, proprietary, and constantly changing, and thus Word tends to end up winning by default, even when writers would prefer to use other, simpler programs. Word 2003 offers the option to save its documents in an XML application called WordML instead of its native binary file format. It is far easier to reverse-engineer an undocumented XML format than a binary format. In the future, Word files will much more easily be exchanged among people using different word processors.

Structured data

XML is ideal for large and complex documents because the data is structured. You specify a vocabulary that defines the elements in the document, and you can specify the relations between elements. For example, if you're putting together a web page of sales contacts, you can require every contact to have a phone number and an e-mail address. If you're inputting data for a database, you can make sure that no fields are missing. You can even provide default values to be used when no data is available.

XML also provides a client-side include mechanism that integrates data from multiple sources and displays it as a single document. (In fact, it provides at least three different ways of doing this, a source of some confusion.) The data can even be rearranged on the fly. Parts of it can be shown or hidden depending on user actions. You'll find this extremely useful when you're working with large information repositories like relational databases.

The Life of an XML Document

XML is, at its root, a document format, a series of rules about what a document looks like. There are two levels of conformity to the XML standard. The first is *well-formedness* and the second is *validity*. Part I of this book shows you how to write well-formed documents. Part II shows you how to write valid documents.

HTML is a document format that is designed for use on the Internet and inside web browsers. XML can certainly be used for that, as this book demonstrates. However, XML is far more broadly applicable. It can be used as a storage format for word processors, as a data interchange format for different programs, as a means of enforcing conformity with intranet templates, and as a way to preserve data in a human-readable fashion.

However, like all data formats, XML needs programs and content before it's useful. It isn't enough to just understand XML itself. That's not much more than a specification for what data should look like. You also need to know how XML documents are edited, how processors read XML documents and pass the information they read on to applications, and what these applications do with that data.

Editors

XML documents are most commonly created with an editor. This might be a basic text editor, such as Notepad or vi, that doesn't really understand XML at all. On the other hand, it might be a completely WYSIWYG editor, such as Adobe FrameMaker, that insulates you almost completely from the details of the underlying XML format. Or it may be a structured editor, such as Visual XML (`http://www.pierlou.com/visxml/`), that displays XML documents as trees. For the most part, the fancy editors aren't very useful as of yet, so this book concentrates on writing raw XML by hand in a text editor.

Other programs can also create XML documents. For example, previous editions of this book included several XML documents whose data came straight out of a FileMaker database. In this case, the data was first entered into the FileMaker database. Next, a FileMaker calculation field converted that data to XML. Finally, an AppleScript program extracted the data from the database and wrote it as an XML file. Similar processes can extract XML from MySQL, Oracle, and other databases by using XML, Perl, Java, PHP, or any convenient language. In general, XML works extremely well with databases.

In any case, the editor or other program creates an XML document. More often than not, this document is an actual file on some computer's hard disk, but it doesn't absolutely have to be. For example, the document might be a record or a field in a database, or it might be a stream of bytes received from a network.

Parsers and processors

An XML parser (also known as an XML processor) reads the document and verifies that the XML it contains is well formed. It may also check that the document is valid, although this test is not required. The exact details of these tests are covered in Part II. If the document passes the tests, the processor converts the document into a tree of elements.

Browsers and other applications

Finally, the parser passes the tree or individual nodes of the tree to the client application. If this application is a web browser such as Mozilla, the browser formats the data and shows it to the user. But other programs may also receive the data. For example, a database might interpret an XML document as input data for new records; a MIDI program might see the document as a sequence of musical notes to play; a spreadsheet program might view the XML as a list of numbers and formulas. XML is extremely flexible and can be used for many different purposes.

The process summarized

To summarize, an XML document is created in an editor. The XML parser reads the document and converts it into a tree of elements. The parser passes the tree to the browser or other application that displays it. Figure 1-1 shows this process.

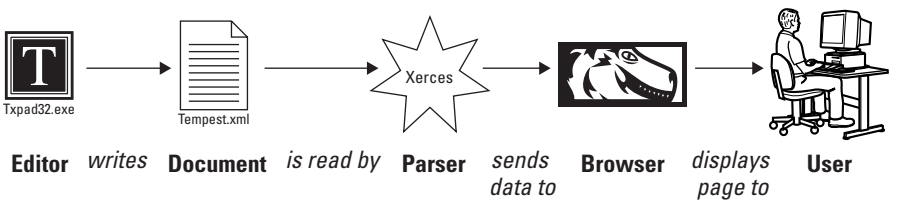


Figure 1-1: XML document life cycle

It's important to note that all of these pieces are independent of and decoupled from each other. The only thing that connects them is the XML document. You can change the editor program independently of the end application. In fact, you may not always know what the end application is. It might be an end user reading your work, it might be a database sucking in data, or it might be something not yet invented. It may even be all of these. The document is independent of the programs that read and write it.

HTML is also somewhat independent of the programs that read and write it, but it's really only suitable for browsing. Other uses, such as database input, are beyond its scope. For example, HTML does not provide a way to force an author to include certain required content such as the ISBN in every book. XML enables you to do this. You can even control the order in which particular elements appear (for example, that level 2 headers must always follow level 1 headers).

Related Technologies

XML doesn't operate in a vacuum. Using XML as more than a data format involves several related technologies and standards, including the following:

- ✦ HTML for backward compatibility with legacy browsers
- ✦ The CSS and XSL style sheet languages to define the appearance of XML documents
- ✦ URLs and URIs to specify the locations of XML documents
- ✦ XLinks to connect XML documents to each other
- ✦ The Unicode character set to encode the text of an XML document

HTML

Mozilla 1.0, Opera 4.0, Internet Explorer 5.0, and Netscape 6.0 and later provide some (albeit incomplete) support for XML. However, it takes about two years before most users have upgraded to a particular release of the software (in 2004, my wife still uses Netscape 4 on her Mac at work), so you're going to need to convert your XML content into classic HTML for some time to come.

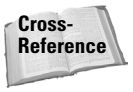
Therefore, before you jump into XML, you should be completely comfortable with HTML. You don't need to be a hotshot graphical designer, but you should know how to link from one page to the next, how to include an image in a document, how to make text bold, and so forth. Because HTML is the most common output format of XML, the more familiar you are with HTML, the easier it will be to create the effects you want.

On the other hand, if you're accustomed to using tables or single-pixel GIFs to arrange objects on a page, or if you begin planning a web site by sketching out its design in Photoshop, you're going to have to unlearn some bad habits. As previously discussed, XML separates the content of a document from the appearance of the document. You develop the content first, and then design a style sheet that formats the content. Separating content from presentation is an extremely effective technique that improves both the content and the appearance of the document. Among other things, it enables authors, programmers, and designers to work more independently of each other. However, it does require a different way of thinking about the design of a web site, and perhaps even the use of different project management techniques when multiple people are involved.

CSS

Because XML allows arbitrary tags in a document, the browser has no way to know in advance how each element should be displayed. When you send a document to a user, you also need to send along a style sheet that tells the browser how to format the tags you've used. One kind of style sheet you can use is a CSS style sheet.

Cascading style sheets, initially invented for HTML, define formatting properties such as font size, font family, font weight, paragraph indentation, paragraph alignment, and other styles that can be applied to particular elements. For example, CSS allows HTML documents to specify that all `H1` elements should be formatted in 32-point, centered, Helvetica bold. Individual styles can be applied to most HTML tags that override the browser's defaults. Multiple style sheets can be applied to a single document, and multiple styles can be applied to a single element. The styles then cascade according to a particular set of rules.



CSS rules and properties are explored in more detail in Chapters 12, 13, and 14.

Mozilla, Opera 4.0, Netscape 6.0, and Internet Explorer 5.0 and later can display XML documents with associated CSS style sheets. They differ a little in how many CSS properties they support and how well they support them.

XSL

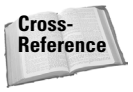
The Extensible Stylesheet Language (XSL) is a more powerful style language designed specifically for XML documents. XSL style sheets are themselves well-formed XML documents. XSL is actually two different XML applications:

- ✦ XSL Transformations (XSLT)
- ✦ XSL Formatting Objects (XSL-FO)

Generally, an XSLT style sheet describes a transformation from an input XML document in one format to an output XML document in another format. That output format can be XSL-FO, but it can also be any other text format (XML or otherwise), such as HTML, plain text, or TeX.

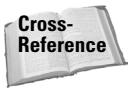
An XSLT style sheet contains templates that match particular patterns of XML elements. An XSLT processor reads an XML document and an XSLT style sheet and compares the elements it finds in the document to the patterns in the style sheet. When the processor recognizes a pattern from the XSLT style sheet in the input XML document, it instantiates the template and outputs the resulting text. Unlike cascading style sheets, this output text is somewhat arbitrary and is not limited to the input text plus formatting information. It depends on the instructions in the template.

A CSS style sheet can only change the format of a particular element, and it can only do so on an element-wide basis. An XSLT style sheet, on the other hand, can rearrange and reorder elements. It can hide some elements and display others. Furthermore, it can choose the style to use based not just on the element name, but also on the contents and attributes of the element, on the position of the element in the document relative to other elements, and on a variety of other criteria.



XSLT is introduced in Chapter 5 and explored in detail in Chapter 15.

XSL-FO is an XML application that describes the layout of a page. It specifies where particular text is placed on the page in relation to other items on the page. It also assigns styles, such as italic, or fonts, such as Arial, to individual items on the page. You can think of XSL-FO as a page description language like PostScript (minus PostScript's built-in, Turing-complete programming language).



XSL-FO is covered in Chapter 16.

Which style sheet language should you choose? CSS has the advantage of broader browser support. However, XSL is far more flexible and powerful, and better suited to XML documents. Furthermore, XML documents with XSLT style sheets can easily be converted to HTML documents with CSS style sheets. XSL-FO is a little past the bleeding edge, however. No browsers support it, and even third-party FO-to-PDF converters such as FOP don't support all of the current formatting object specification.

Which language you pick largely depends on your use case. If you want to serve XML files directly to clients and use their CPU power to format and transform the documents, you really need to be using CSS (and even then, the clients had better have very up-to-date browsers). On the other hand, if you want to support older browsers, you're better off converting documents to HTML on the server using XSLT, and sending the browsers pure HTML. For high-quality printing, you're better off with XSLT plus XSL-FO. An advantage of XML is that it's quite easy to do all of this at the same time. You can change the style sheet and even the style sheet language you use without changing the XML documents that contain your content.

URLs and URIs

XML documents can live on the Web, just like HTML and other documents. When they do, they are referred to by Uniform Resource Locators (URLs). For example, at the URL `http://cafeconleche.org/examples/shakespeare/tempest.xml` you'll find the complete text of Shakespeare's *Tempest* marked up in XML.

Although URLs are well understood and well supported, the XML specification uses the more general Uniform Resource Identifier (URI). URIs are a more general scheme for locating resources; URIs focus a little more on the resource and a little less on the location. Furthermore, they aren't necessarily limited to resources on the Internet. For example, the URI for this book is `urn:isbn:0764549863`. This doesn't refer to the specific copy you're holding in your hands. It refers to the almost-Platonic form of the third edition of the *XML Bible* shared by all individual copies.

In theory, a URI can find the closest copy of a mirrored document or locate a document that has been moved from one site to another. In practice, URIs are still an area of active research, and the only kinds of URIs that current software actually supports are URLs.

XLinks and XPointers

As long as XML documents are posted on the Internet, people will want to link them to each other. Standard HTML link tags can be used in XML documents, and HTML documents can link to XML documents. For example, this HTML link points to the aforementioned copy of the *Tempest* in XML:

```
<A HREF=
"http://cafeconleche.org/examples/shakespeare/tempest.xml">
  The Tempest by Shakespeare
</A>
```

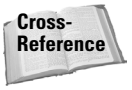
Whether the browser can display this document if you follow the link depends on just how well the browser handles XML files. Fourth-generation and earlier browsers don't handle them very well.

However, XML lets you go further with XLinks for linking to documents and XPointers for addressing individual parts of a document.

XLinks enable any element to become a link, not just an `A` element. For example, in XML, the preceding link might be written like this:

```
<PLAY xlink:type="simple"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xlink:href=
   "http://cafeconleche.org/examples/shakespeare/tempest.xml">
  <TITLE>The Tempest</TITLE> by <AUTHOR>Shakespeare</AUTHOR>
</PLAY>
```

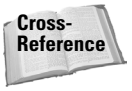
Furthermore, XLinks can be bidirectional, multidirectional, or even point-to-multiple mirror sites from which the nearest is selected. XLinks use normal URLs to identify the site to which they're linking. As new URI schemes become available, XLinks will be able to use those, too.



XLinks are discussed in Chapter 17.

XPointers allow links to point not just to a particular document at a particular location, but to a particular part of a particular document. An XPointer can refer to a particular element of a document; to the first, the second, or the seventeenth such element; to the first element that's a child of a given element; and so on. XPointers provide extremely powerful connections between documents that do not require the targeted document to contain additional markup just so its individual pieces can be linked to another document.

Furthermore, unlike HTML anchors, XPointers don't just refer to a point in a document. They can point to ranges or spans. For example, an XPointer might be used to select a particular part of a document so that it can be copied or loaded into a program.



XPointers are discussed in Chapter 18.

Unicode

The Web is international, yet a disproportionate amount of the text you'll find on it is in English. XML is helping to change that. XML provides full support for the Unicode character set. This character set supports almost every character that is commonly used in every modern script on Earth.

Unfortunately, XML and Unicode alone are not enough to enable you to read and write Russian, Arabic, Chinese, and other languages written in non-Roman scripts. To read and write a language on your computer, it needs three things:

1. A character set for the script in which the language is written
2. A font for the character set
3. An operating system and application software that understand the character set

If you want to write in the script as well as read it, you'll also need an input method for the script. However, XML defines character references that allow you to use pure ASCII to encode characters not available in your native character set. This is sufficient for an occasional quote in Greek or Chinese, although you wouldn't want to rely on it to write a novel in another language.

Putting the pieces together

XML defines the syntax for the tags you use to mark up a document. An XML document is marked up with XML tags. The default character set for XML documents is Unicode.

Among other things, an XML document may contain hypertext links to other documents and resources. These links are created according to the XLink specification. XLinks identify the documents that they're linking to with URIs (in theory) or URLs (in practice). An XLink may further specify the individual part of a document it's linking to. These parts are addressed via XPointers.

If an XML document is intended to be read by human beings — and not all XML documents are — a style sheet provides instructions about how individual elements are formatted. The style sheet may be written in any of several style sheet languages. CSS and XSL are the two most popular style sheet languages, and the two best suited to use with XML.

Summary

In this chapter, you've seen a high-level overview of what XML is and what it can do for you. In particular, you learned the following:

- ✦ XML is a meta-markup language that enables the creation of markup languages for particular documents and domains.
- ✦ XML tags describe the structure and semantics of a document's content, not the format of the content. The format is described in a separate style sheet.
- ✦ XML documents are created in an editor, read by a parser, and displayed by a browser.
- ✦ XML on the Web rests on the foundations provided by HTML, CSS, and URLs.
- ✦ Numerous supporting technologies layer on top of XML, including XSL style sheets, XLinks, and XPointers. These let you do more than you can accomplish with just CSS and URLs.

The next chapter presents a number of XML applications that demonstrate the ways that XML is being used in the real world. Examples include vector graphics, musical notation, mathematics, chemistry, human resources, and more.

✦ ✦ ✦